Karl Moore's Visual Basic .NET: The Tutorials

KARL MOORE

Apress™

Karl Moore's Visual Basic .NET: The Tutorials
Copyright © 2002 by Karl Moore

ISBN (pbk): 1-59059-021-X

Printed and bound in the United States of America 12345678910

Technical Reviewers: Patricia Moore, Richard Costall, Ray Ellison, Erik Giggey, Donald Carter
Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore, Karen Watterson, John Zukowski
Managing Editor: Grace Wong
Project Manager: Alexa Stuart
Copy Editor: Tom Gillen
Production Editors: Janet Vail and Grace Wong
Compositor: Impressions Book and Journal Services, Inc.
Indexer: Ann Rogers
Cover Designer: Tom Debolski
Marketing Manager: Stephanie Rodriguez

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010
and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.
In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com.
Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth St., Suite 219, Berkeley, CA 94710.
Phone: 510-549-5903, Fax: 510-549-5933, Email: info@apress.com, Web site: http://www.apress.com

The source code for this book is available to readers at http://www.apress.com in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

TUTORIAL 6

Services Rendered

6.1

Introducing the World of Web Services

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."—Richard Cook

TRUTH BE KNOWN, COMPUTERS ARE highly unsociable creatures.

I mean, they won't even talk to each other unless you string a mass of cable between them. And, if you're actually looking to get them engaged in any sort of productive conversation, you'll need to tweak network settings, twist hub knobs, pray nightly to the almighty power of Bill Gates, *and* cross your fingers.

Thankfully, however, there's a cure for such typically timid devices. It's the .NET Framework, and it works like alcohol for computer chips. Install this baby and suddenly your quiet chunk of silicon seems better at schmoozing than Dale Carnegie.

Why? Because that .NET Framework includes ASP.NET. And ASP.NET includes Web services. And Web services give you the ability to write programs on different machines that can talk to each other, easily.

Your client program might ask your main computer for information about a customer, for example. Or query the government for its latest tax rates, say. Or nab late-breaking headlines straight from CNN. Or *whatever*.

It's all possible, when you get talking—via Web services.

Could be useful? Fancy learning more? Then sign up and step this way . . .

How It All Works

Sometimes you simply can't do everything on one computer. It's often best to separate the functionality, using programs that talk to and help applications on different computers. Enter stage left: *Web services*.

Now, imagine you work for Deals on Wheels, the most popular car dealership this side of the Mississippi. The whole group runs on your applications, and, with five popular sites, you're exceptionally pleased.

The problem is your users aren't. And, in stereotypical user fashion, they do nothing but moan and request new features. (Difficult to imagine, I know.) Anyway, first on their list of improvements is an automatic credit check performed on each client as soon as their details are entered into your system.

Oh, great. I mean, how on Earth can you implement something like that? Hmm, I guess you could just write the credit-checking code yourself, then update all the users with a new version of your customer database program. But surely that wouldn't be quite so comprehensive as a third-party check, and what about that "credit check free" day they were talking about? Yuck. What an administrative nightmare.

Actually, maybe it's just easier to tell the users that they should physically call your checking agency. After all, *they've* got the information, and *your* computers can't talk to *theirs*, can they?

What's that? Oh, your boss Dodgy Del has his own personal request, too? When a customer steps through his doors to trade in a rusty old motor, Sam the Salesman instantly heads to his desk and checks out the trade-in value in his pricing guide. Problem is, this is often the *1986* pricing guide—and Del guesses he's losing thousands of dollars a month due to out-of-date information.

He suggests a centralized database containing all the latest prices—and you shiver, thinking of all the different applications you'll need to update for this. Reams of data access code in three different languages flash before your eyes. But, after all, there's no easier way, is there?

Hmm, I could just say "no" to both of these questions, but then that'd make for a real short chapter. Instead, I'm going to answer "yes," and welcome you to the world of Web services, part of ASP.NET. Erm, so just what are Web services?

Web services are a way of creating programs that talk to and help applications on different computers. You have a server machine that "exposes" the service and clients that "discover" and "consume" that service.

So, your credit-checking agency may make a service available over the Internet that allows you to pass details of your customer to it in code, and the service will return the credit rating. Or you might create your own service on the main company server that looks up car prices in a database, and then simply call this one service from each of your applications. That's what Web services allow you to do: get programs on different computers talking together. They allow you to *distribute* the functionality of your application. How could something like this help you, right now?

Hmm, you still look a little suspicious. I'm guessing you think I've slipped off into my own little fantasy world once again and that this sounds all too good to be true. Well, I've had my cold shower, and can inform you that it *is* possible, right now, today, with Web services.

How? Well, you (or perhaps that credit-checking agency) start by creating a Web service project in Visual Studio .NET. This is basically just a project containing code, a little like the class library we worked with in Tutorial 5, "Using Objects". Then the development begins, and our friendly coder tells VB .NET to show certain bits of his code to the world.

So, maybe he'll tell it to expose that DoCreditCheck method or perhaps that GetCarPrice function.

When finished, the code is compiled, and that "Web service" is made available on a master computer somewhere—our network, the Internet, or wherever. And that's it for your Web service: it's up, running, and waiting to serve.

Next, it's time to develop your client, the program that will use the Web service. You start by "discovering" the Web service and its features, then use all those exposed functions exactly as you would a local piece of code, and they run, all the way over on the other machine. It all just works: you do nothing special.

The Geeky Bit

That's right: *you* do nothing special. However, our lovely .NET Framework is being smarter than da Vinci in Gucci. When you run the functions belonging to that Web service, it calls the remote machine passing any parameters, runs the Web service code, and returns any data back to your client program *automatically*. Complete doddle.

And all of this works via something called SOAP. Shower jokes aside, *SOAP* stands for the *Simple Object Access Protocol* and refers to how Web services talk. And how do they talk? Behind the scenes, everything is converted and passed about as XML, which is just plain, structured text. And this XML is transferred over HTTP, which is typically used for viewing Web pages. (See Figure 1-1.)

Why is this clever? Two reasons. First off, XML is pure text, and pure text is *speedy* and *understood by all platforms*. And, because it works via HTTP, it also bypasses any annoying firewalls that may attempt to thwart your development efforts. But, again, this is all handled automatically for you, so there's no need for concern. Again, it's just a complete doddle.

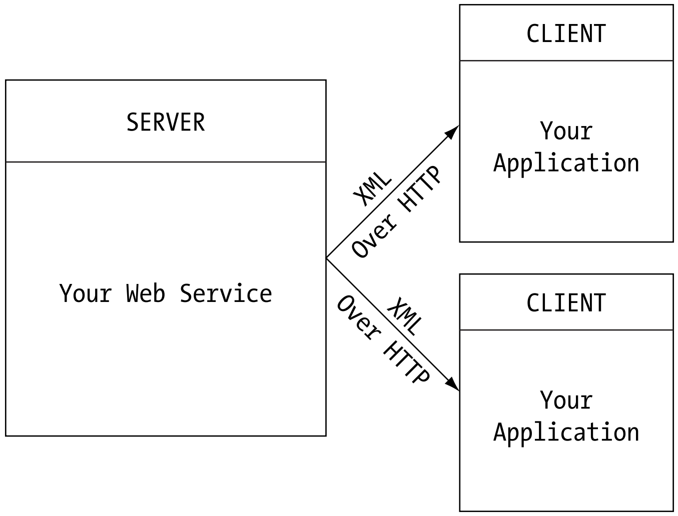


Figure 1-1. A Web service in action

And if you're coming from the old school of DCOM and all that jazz, you're in for a particularly pleasant surprise: with Web services there's no registering, no settings to fine-tune, no annoying tight-coupling. Basically, it's a complete doddle. Did I mention that?

So, Web services allow you to get computers talking. You expose certain functionality through a service, and your clients "discover" and "consume" that service, live via XML over HTTP.

Ready to start exploring Web services in real life? Let's dive into meat space and put all this fantasy into practice. Oooh, matron!

Creating a Web Service

George Bush (Sr.) once said, "I pushed the button down here and one up here with the green thing on it. And out came a command to somebody that I had written."

Uh-huh. Well, perhaps creating Web services wasn't for him. But, for us, it's time to put all this theory into action, getting our palms dirty with a little hands-on.

1. Launch Visual Studio .NET and create a new ASP.NET Web service, changing its location to `http://localhost/MyHelper`.

After a little whizzing and whirring, you should be presented with a pretty dull-looking screen. This is Service1.asmx.vb, the code file behind your actual service. However, I don't see any code right now, presumably because we're in design mode. Let's change that.

2. On the menu, select View ➢ Code to open the code window.

TOP TIP *Instead of constantly using the menu to open the code window, just press F7. It'll take you straight to the code behind the object you're working with.*

You should see a small mound of code in front of you, although nothing shockingly horrid. We have a line that imports the Web service namespace, making its functionality available to use. Aside from that, it just looks like a regular class that inherits from the WebService class. It also includes a bundle of comments, currently wearing a very chic shade of green.

Now, let's add a function to this class—just a bog-standard, run-of-the-mill function.

3. Add the following function to your class:

```
Public Function Reverse(ByVal Text As String) As String
     Return StrReverse(Text)
End Function
```

Well, it beats a simple Hello World sample, but only *just*. Yes, it's a chunk of code that takes a string and reverses it, but, unless you're about as bright as a blown bulb, I guess you already figured that out.

Anyway, now it's time to reveal the real secret of Web services. It's the key to instantly exposing your data to the world, it's the backbone to the entire .NET concept, it's . . . `<WebMethod()>`.

Yes, I know. Profound. But just by adding this simple "attribute," we're telling VB .NET that this is a "WebMethod," a method or function that should be made available via your network, the Internet, or wherever.

4. Add the `<WebMethod()>` attribute to the beginning of your function, so it reads:

```
<WebMethod()> Public Function Reverse(ByVal Text As String) As String
    Return StrReverse(Text)
End Function
```

And surprise, surprise—that's our Web service finished! Let's build our solution, then see what this simple `<WebMethod()>` attribute has done for us.

5. From the menu, select Build ➢ Build Solution (or press `Ctrl+Shift+B`).

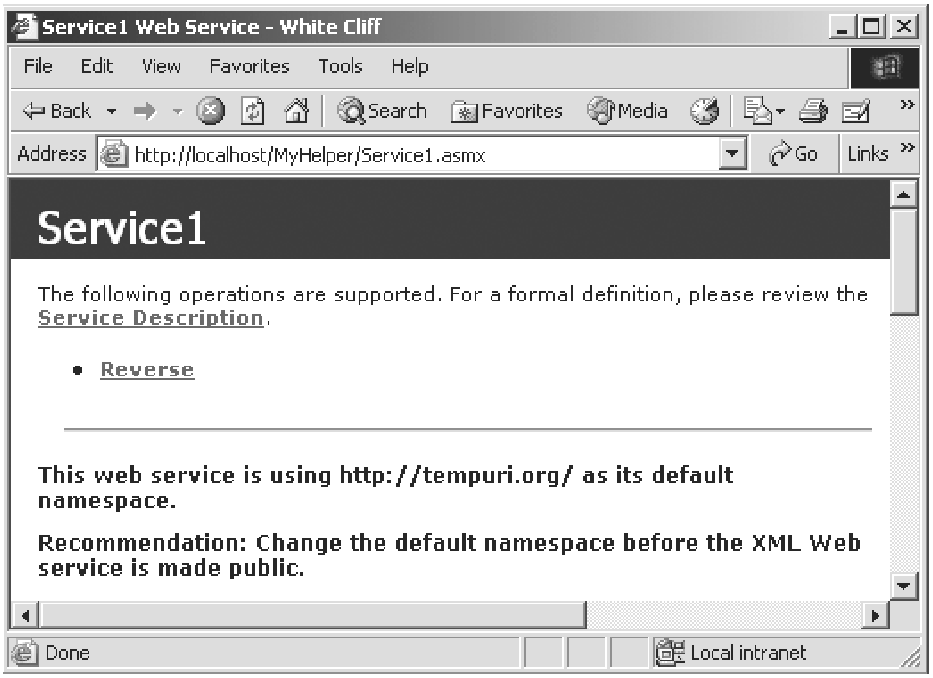
Accessing the Web Service from a Browser

There are two key ways to access your Web service. The first is from a Web browser. That's our cue.

1. In your application, press F5 to start your Web service.

Your default browser should fire up, directing itself to somewhere like `http://localhost/MyHelper/Service1.asmx`.

What you're looking at in Figure 1-2 is an automatically generated interface to your HTTP Web service.


Service1 Web Service - White Cliff
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/MyHelper/Service1.asmx
Go Links
Done
Local intranet

Service1

The following operations are supported. For a formal definition, please review the Service Description.

- Reverse

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Figure 1-2. Accessing our Service from a browser

Here, we can see our Web service is very originally entitled "Service1" and that it exposes one operation called "Reverse".

2. Click on the Reverse link.

You'll be taken to a page that describes our Reverse operation. From here, you can either test it directly or check out some simply *thrilling* sample SOAP code. Think I'll just test it.

3. Enter a value for the Text parameter, such as your name.

4. Click on the Invoke button to run your function.

A separate window should pop open and display something like:

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">erooM lraK</string>
```

Huh? Well, this is XML, that "eXtensible Markup Language" we talked about earlier and stumbled across back in Tutorial 2, "Doing Databases". Here, this chunk of XML is telling us that our Reverse operation has returned a string saying "erooM lraK". Think I'll just stick to *Karl*.

> **TOP TIP** *XML is the glue that holds a lot of .NET together. It's essentially a self-describing structure that can hold hierarchical data. Or at least that's the official lowdown. Personally, I just think, "Hmm, looks a bit like HTML." Simple minded, you see.*

5. Close your browser windows and click on the Stop button in VB .NET.

So, that's a Web service. Okay, admit it: it looks about as useful as an ice machine at the North Pole. But this is only the raw Web interface. There's another, much more powerful way of accessing them, a way that completely hides all the XML from us and makes development easy.

I'm talking about using Web services . . . in your applications.

Accessing the Web Service from an Application

Next, let's throw together an application that actually "consumes" this Web service. You can create this on the same machine as your Web service if you like, or on another computer in your network. It really doesn't matter; as long as your browser can access the Web service, your program will be able to also.

1. Create a new Windows application using Visual Studio .NET.

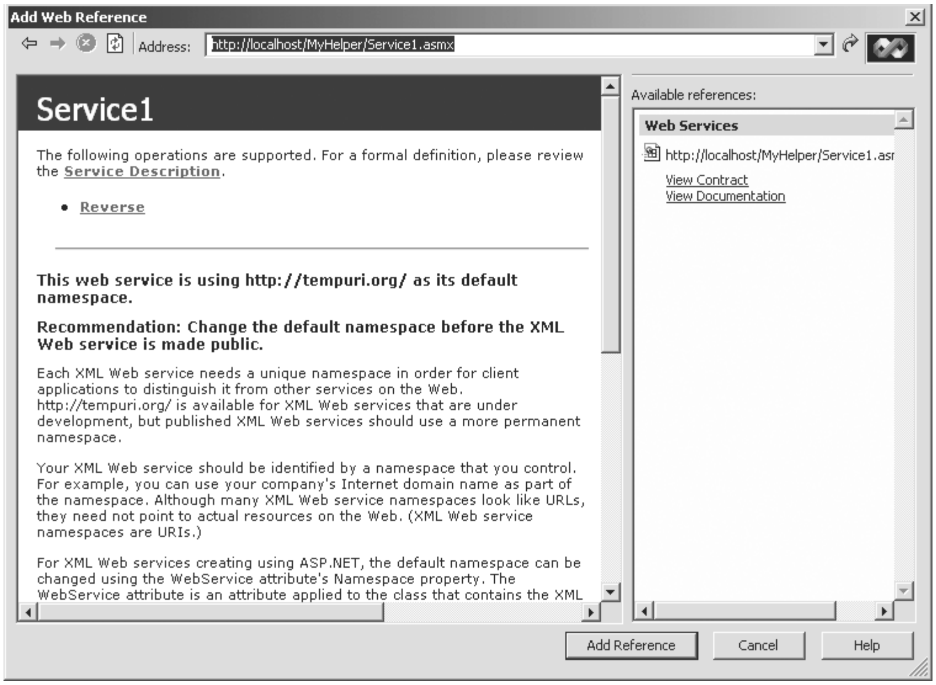
2. From the menu, select Project ➢ Add Web Reference.

You should be looking at the Add Web Reference screen. If not, either your machine has contracted the Squiggle virus, or you've accidentally unplugged your monitor. Next, you need to tell VS .NET where your Web service is located.

3. Type in your Web service ASMX address, which will be something like `http://localhost/MyHelper/Service1.asmx`, and press the Enter key.

TOP TIP *If your Web service is on the local machine, you can try clicking on the "Web References on Local Web Server" link to locate your service. If your service is on another machine in the network, replace the "localhost" portion of the sample address with the actual machine name, IP address, or Web address—basically, whatever you'd type into a Web browser to access your Web service.*

4. When your Web service appears, click the Add Reference button. (See Figure 1-3.)



http://localhost/MyHelper/Service1.asmx

Service1

The following operations are supported. For a formal definition, please review the **Service Description**.

- **Reverse**

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the XML

Figure 1-3. Adding a reference to our Web service

Take a glance at the Solution Explorer. You'll notice a new Web References entry there, with "localhost" or your machine name just underneath. Try expanding this node, browsing all the files underneath it. These have all been automatically generated and exist to tell VB .NET how to talk to the service on that machine.

5. Draw a button out onto the Form designer.

6. Behind the click event of the button, add the following code:

```
Dim objHelper As New localhost.Service1()
```

Here, we're treating our Web service just like any regular object we've seen so far in this book. Once again, you may need to change "localhost" to your machine name, as listed in the Solution Explorer.

TOP TIP *Our code here is actually working with a "helper" class that Visual Studio .NET automatically creates for you. When you use its properties and such, this "wrapper" class makes the Web service call, processes the results, and passes them back to your code. If you want to look at the Helper class code, click on "Show All Files" in the Solution Explorer, then expand the Web References folder, then localhost. For each Web reference here, you'll see two XML-based files: a DISCO file that provides details on discovering the service, and a WSDL (Web Service Discovery Language) file that provides details on the capabilities of the service. Expand that ServiceName.WSDL file and look at the ServiceName.VB file underneath it. This is the Helper class code. You can even edit it, if you're feeling particularly randy.*

That's our new objHelper object instantiated. Now we're ready to use it. Try typing `objHelper` and pressing the period key immediately after. What happens? (See Figure 1-4.)

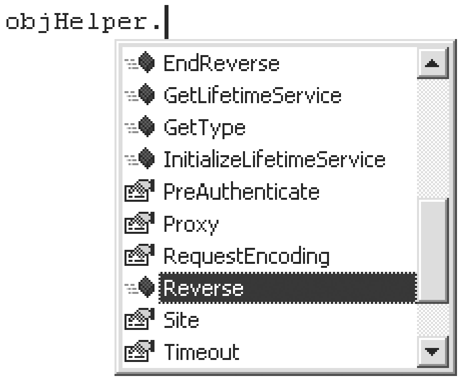


Figure 1-4. Our Web service in the code window

What's that? You don't remember coding all those features? Well, you're either a Steve Jobs reincarnate who can program subconsciously (and at rather an impressive pace), or these are simply standard extras above and beyond your own simple function, extra widgets automatically added to help you use the Web service. I might be wrong, but I'd go for option two.

Let's use our Reverse function in code now. Here, we're simply going to display a message box, telling VB to display the result of Reverse on "Karl Moore". (That's me, by the way.)

7. Underneath your existing button click code, add the following code:

```
MessageBox.Show(objHelper.Reverse("Karl Moore"))
objHelper = Nothing
```

Understand? We're just telling it to run our function with this particular string and to display the results in a message box. Finally, we just set our objHelper variable to nothing, telling it that we're no longer using it: we're just being tidy, as learned how back in Tutorial 5, "Using Objects".

8. Press F5 to test your program.

9. Click on your button.

What happens? After a short delay, you should get back “erooM lraK”.

Now try clicking on the button a second time. See what happens? First time round it was about as slow as a slug. To be specific, a dead slug. Now it zaps away at the speed of light, having already made that initial connection. Just a consideration.

TOP TIP *Wondering how to discover and consume a Web service in something other than a Windows application? Perhaps you want to use the functionality from an ASP.NET Web application or maybe even another Web service? No worries: it all works in exactly the same way. Just “Add Web Reference” and start coding. There’s no difference.*

If you’ve got a few minutes before moving on, try creating a function that returns an integer as opposed to a string. How does the XML differ? Also, can you use multiple parameters just as you can with a regular function? One more test: try adding a method or function that doesn’t include that WebMethod attribute. What happens? Can you see it via your browser?

Getting bored yet? No? Oh, well, let me tell you about my Victorian stamp collection. Oh, you *are* getting bored? Right then, let’s move on and imagine some of the possibilities here.

Imagining the Possibilities

What’s that? The complexity of your organization slightly surpasses our Reverse sample? Oh, darn. Sorry about that. Maybe I can offer you a coupon against my next book or something.

But, you see, although we’ve covered only the core essentials here—and pretty quickly, too—it’s still more than enough for you to get your own Web services into production. It’s all about `<WebMethod()>`.

Just imagine the possibilities open to you, right now. Imagine looking up car trade-in prices from any of your applications, simply by using your one centralized function. Imagine adding new customers just by calling an Add method on your Customer Web service. Imagine being able to change any part of your application code just by editing one project on your server.

<takes deep breath>

Imagine consuming a third-party, Internet-based Web service to authorize a MasterCard transaction or to check a customer’s credit. Imagine exposing your own product stock information over the Web, or making data such as weather reports and sports scores available to your service subscribers.

TOP TIP *The UDDI Directory from Microsoft attempts to list many of the third-party Web services available for you to consume. You can view the directory by visiting* `http://uddi.microsoft.com/visualstudio` *in the Add Web Reference dialog box. Alternatively, search it by clicking the XML Web services link on the Visual Studio .NET start page. Another useful directory can be found at* `http://www.xmethods.com`.

At the time of going to press, among the public Web services in action were a live dictionary and thesaurus, a horoscope generator, an SMS service, an NFC headlines application, and a music teacher search engine. Diverse? Oh yes. *Just imagine.*

And you think that's all? No, sir. You see, with all these possibilities, passing about plain data is often not enough. You need to crank up the volume, with *smart data*—and we'll see at how you can do that, next.

Passing Smart Data

The problem with using Web services as we've shown here is that they're a little limiting. I mean, let's say that you've created a groovy customer management Web service on your network. Perhaps it contains an Add method that accepts details of your customer and adds them to the database. No problem. But what about actually *retrieving* your customers?

Think about it. Functions can return only one item, such as a True or False, or perhaps a string. They can't, for example, return a list of customers and their orders . . . unless, of course, you pass around *smart data*.

I'm talking about relational, multitable DataSets. I'm talking about arrays. I'm talking about classes. I'm talking about structures. I'm talking about smart data, data containing multiple portions that can be passed back through a function—even in a Web service.

To demonstrate the concept, I'd like to create a Web service that opens a database of your choice, then passes all the required data from one particular table data back to your user as an easy-to-use, typed DataSet. Okay, let's get started.

1. Create a new ASP.NET Web service at `http://localhost/MyDataService`.

Now, we're going to start off by using the Server Explorer to connect to your database. I'm going to leave this bit up to you; you can connect to your company database, one of the sample Access or SQL Server databases we created back in Tutorial 2, "Doing Databases", or perhaps just infamous Northwind. Your call.

Personally, I'm going to connect into the Surgery.mdb sample we created back in Tutorial 2, "Doing Databases".

2. From the menu, select View ➢ Server Explorer.

3. If your database is not shown under Data Connections or as an SQL Server database under the Servers entry, click on the "Connect To Database" button and enter the details. (See Figure 1-5.)

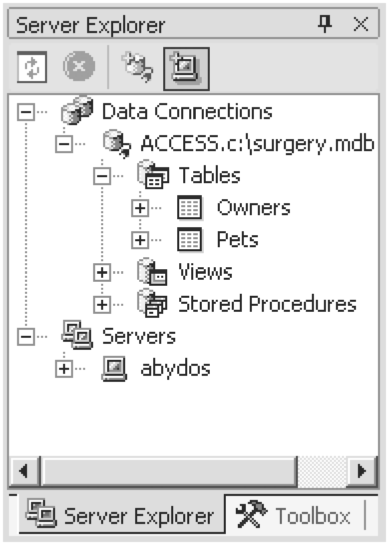


Figure 1-5. Browsing a database via the Server Explorer

4. Expand the Tables list under your database.

5. Drag and drop the table you want to use in this example onto your Service1.asmx.vb page (currently in design mode).

You'll find OleDbConnection1 and OleDbDataAdapter1 added to your service—or perhaps SqlConnection1 and SqlDataAdapter1 if you're using SQL Server. As you may remember from Tutorial 2, "Doing Databases" (*please* say you've read it), the Connection object here actually "calls" your database, and the DataAdapter talks through that connection to get information out or to modify existing data.

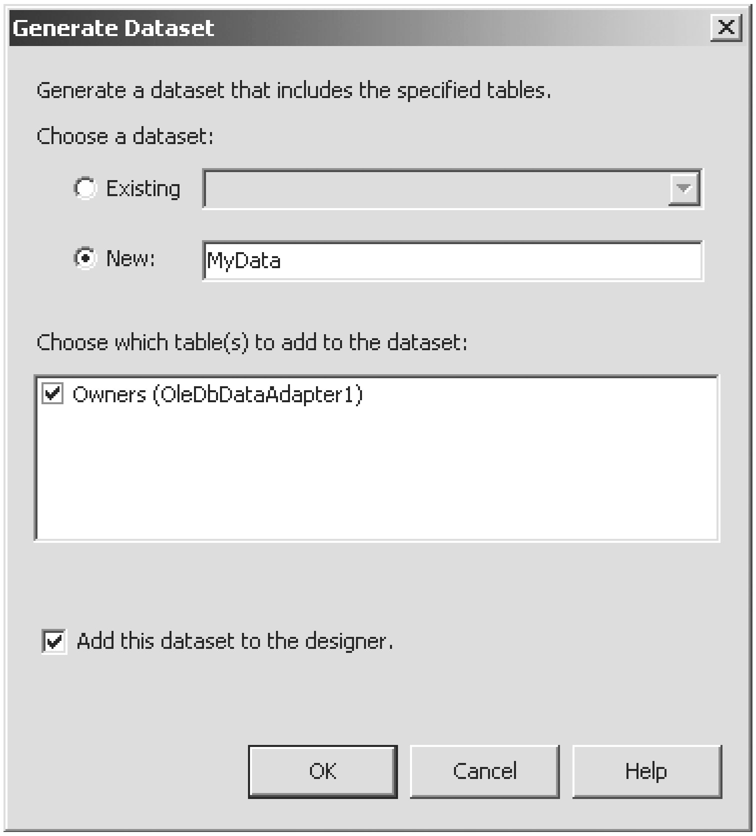
6. Rename your Connection object to "connMyDatabase".

7. Rename your DataAdapter object to "daMyDataAdapter".

Next, let's sort out our DataSet. Eh? A *DataSet* is basically a widget that holds the data coming back from your database. Actually, we're going to create a *Typed DataSet*, which is just a regular DataSet that runs off a template telling it about the tables and fields it will hold, thereby making our later programming much easier.

8. Right-click on your DataAdapter object and select Generate DataSet.
9. Choose the New option and enter a name of "MyData".
10. Ensure your table is checked in the list of tables to add to this DataSet.
11. Check the "Add This DataSet To The Designer" box.
12. Click on OK.

Two things just happened here. First off, MyData.xsd has been added to your project; it lists the fields that your DataSet will hold. Secondly, MyData1 has been added to your Web service. This is your typed DataSet, based on that MyData.xsd "template." (See Figure 1-6.)


Generate Dataset

Generate a dataset that includes the specified tables.

Choose a dataset:

( ) Existing

(•) New: MyData

Choose which table(s) to add to the dataset:

[✓] Owners (OleDbDataAdapter1)

[✓] Add this dataset to the designer.

Figure 1-6. Generating our typed DataSet

13. Rename MyData1 to “dsMyDataSet”.

Great! We’ve sorted the data access objects out. Next, we need to go add code to our Web service to retrieve information from this database and pass it back.

14. Press F7 to switch to the code mode for your Web service.

15. Add the following code to your Service1 class:

```
    <WebMethod()> Public Function GetData() As MyData
        ' Passes populated Typed DataSet back to client
        daMyDataAdapter.Fill(dsMyDataSet)
        Return dsMyDataSet
End Function
```

Here we have a function called GetData that returns a DataSet of the MyData type, a DataSet that adheres to our MyData “template” class. The code simply fills our DataSet with information from our database table and returns it. Simple.

Oh, and of course we have that magical keyword `<WebMethod()>` there, too. Leave it out and this service can’t strut its stuff.

However, most databases aren’t read-only: you’ll typically want to update them, too. Next, we’re going to accept the entire DataSet back and update the backend database as required, However, in the real world, you might just want to accept individual Row objects and go from there.

16. Add the following code to your Service1 class:

```
<WebMethod()> Public Sub UpdateData(ByVal Data As MyData)
     ' Updates the backend database where needed
     daMyDataAdapter.Update(Data)
End Sub
```

Here, we’re just accepting a MyData typed DataSet and using the DataAdapter object to update our backend database. No problem. Well, that’s our whole Web service pretty much finished now and our cue to run a quick build.

17. Press `Ctrl+Shift+B` to build this solution.

Retrieving Smart Data

But what good is data without someone to use it? As Confucius once said, "It ain't." Which is why we're about to create a neat little Windows application to discover and consume the features of our new service.

1. Create a new Windows application.

2. From the menu, select Project ➢ Add Web Reference.

3. Type in your Web service address:
   `http://localhost/MyDataService/Service1.asmx`.

4. Click on "Add Reference" when the page has loaded.

Next, let's add code to test our Web service.

5. Add a button to Form1.

6. Behind the click event of your button, add the following code:

```
' Declare Service and DataSet 'template'
Dim MyWebService As New localhost.Service1()
Dim MyDataSet As localhost.MyData

' Grab data
MyDataSet = MyWebService.GetData

' Retrieve sample data -
' your fields may be different
Dim strText As String
strText = MyDataSet.Owners(0).Address
MessageBox.Show(strText)

' Update DataSet –
' again, your fields may be different
MyDataSet.Owners(0).Name = "Mr Bibbles"

' Send back to Web service
MyWebService.UpdateData(MyDataSet)
```

Okay, it might be a little more in-depth than our Reverse sample, but it's still pretty understandable. We're creating a new instance of our own local Service1 class, then using its functionality to work with our Web service—grabbing our data, displaying one particular field, changing a little data, then running an update. Top stuff.

The special thing to note here is that, when we compiled our Web service, it recognized we were exposing a typed DataSet and included details of our MyData schema alongside, allowing us to use its objects in code as we have done here. How very clever.

7. Press F5 and test your application.

Does it all work as expected? How could we improve the interface? Well done on completing this sample!

So, what have we created in under two dozen lines of code? On the server side: a Web service that exposes data direct from your database using a typed DataSet. On the client side: an application that can use all the power of your "disconnected" data in just a few lines of code—without having to worry about the data access code in the slightest.

This is smart data. It's a lot more than just a single True or False return value, and its simplicity can really help speed up your application development. And what do you know—it doesn't stop with DataSets. You can expose data-holding classes, structures, and arrays through your Web services, too. Just use them and let Visual Studio .NET do the rest.

Tsk, can you spell *clever chuff*?

The Top Five Distribution Tips

So, you've written the greatest Web service since SlicedBread.asmx, and now you want to roll it out to the world? Yes, you could just take what you've done so far and follow the deployment instructions in the next section, but five quick changes will give your service that professional touch, mini alterations that should precede the distribution of *any* Web service.

Get out that notebook. It's time to talk turkey.

1. Rename your class.

Nobody likes dealing with a Web service called Service1, so, before you redistribute, make sure that you rename your class from something like `Public Class Service1` to something like `Public Class Customer`.
Or something. Ahem.

2. Christen your ASMX file.

You've renamed your Web service so it sounds all professional, but your clients are still accessing your Service via the `http://localhost/Customer/Service1.asmx` file. Eugh. To combat this, simple rename your ASMX file by right-clicking on it in the Solution Explorer and selecting Rename.

3. Add a service description.

What does your Web service do? Provide access to your customer database or allow you to send SMS messages? Well, you could just let your clients guess. However, a more user-friendly technique would be to add a description to the `<WebService()>` attribute. So, before distributing, change your class as follows:

```
<WebService(Description:="This is a class that does amazing things with data.")> _
 Public Class AmazingData
   ' . . . etc . . .
 End Class
```

This description is used in the automatically generated Web interface and is automatically absorbed by all clients using your service and utilized in the Visual Studio .NET programming environment.

4. Add method descriptions.

Just as your service can include a description, so can your individual methods and functions. Again, this is used by the Web interface and Visual Studio .NET. And, again, it's simple to implement: just add a description to each `<WebMethod()>` attribute, like so:

```
<WebMethod(Description:="Returns the current server date and time.")> _
 Public Function GetServerDate() As Date
     Return Now
 End Function
```

5. Change the namespace.

You've already seen the .NET Framework organizing functionality into "namespaces." They're just unique identifiers for a certain set of functionality. That's all a namespace is here, too: a unique string that identifies this Web service. By default, it's `http://www.tempuri.org/`, and you will need to change this to a unique value. The namespace you provide doesn't necessarily have to point to anything on the Web, but it's recommended that you at least own the domain you use (I mean, *purr-lease*). So, change the namespace to something unique before unveiling your service by altering the Namespace property of the `<WebService()>` attribute, as so:

```
<WebService(Description:="Yadda",  _
Namespace:="http://www.amazingdata.com/query/")> _
 Public Class AmazingData
  ' . . . etc . . .
 End Class
```

Deploying Your Web Service

Once you've finished your Web service, you can stick it literally anywhere. And, resisting another crude Bill Gates prod, I'm talking about the Internet or your network.

How? Well, if you built your Web service on the server it will be running on, you don't actually need to do anything else. When you build your service in Visual Studio .NET, it's automatically deployed for you.

However, if you want to move it over to another machine, you'll need to use the Copy Project feature. Simply open your Web service in Visual Studio .NET and select Project ➢ Copy Project from the menu. Change the Project Destination folder to the new HTTP home of your service and ensure that the "Copy: Only files needed to run this application" option is selected. Then click on the OK button.

TOP TIP *You've seen that Web services run over HTTP. In fact, they work in just the same way as regular ASP.NET Web applications, which means that you can administer them via the Internet Services Manager plug-in: to launch it, select Start ➢ Program Files ➢ Administrative Tools ➢ Internet Services Manager. Useful for deleting all those Hello World samples!*

Alternatively, if you're deploying via FTP to a .NET host, copy across your ASMX pages, Web.config and VSDISCO files. You'll also want to transfer your compiled DLL assembly from your local Bin subdirectory to a Bin subdirectory on the server. Basically, copy across anything that isn't a Visual Studio .NET project file, in the exact same way as you would with an ASP.NET Web application.

You can also upload direct to a supported .NET host by clicking the Web Hosting link on the Visual Studio .NET Start page.

And that's a rap, folks: your Web service is now exposed to the world. Let the consuming commence!

FAQ

Still got a couple of questions about Web services? Looking to learn from the mistakes of others? Let's review our list of frequently asked questions . . .

In your examples, you've worked on a Web service project with just one Service class. Can you have multiple Service classes in one Web service project?

Absolutely! Simply add another Web service to the project, by selecting Project ➢ Add Web Service from the menu. You can even add a Web service to a regular ASP.NET Web application in exactly the same way.

My code always seems to have something wrong with it. Is there any way to debug a Web service, just as you would a Web form or a regular Windows application?

Yes—literally *just* as you would a Web form or regular Windows application! Simply move to the line you wish to break on and press the F9 key to turn debugging on. Next, run your service live by pressing F5—then either run the application that uses this Web service or access its operations via the Web interface. When the line you highlighted is about to run, your code will break and you should be able to step through line by line using F8. Happy bug hunting!

What happened to all those ASP.NET objects, like Application and Session?

Well, because Web services are a part of ASP.NET, objects such as Application, Session, Server, User, and all their merry friends are still at your disposal (though admittedly not used here quite as often). Check out the Help index or Tutorial 3, "Working the Web" to find out more.

I've added a few new methods to my Web service. Have I now got to update all my clients?

Don't worry—as long as you haven't "broken" any of your existing methods, you won't have any problems. If, however, you want to use these new methods in your applications, open the client project and, using the Solution Explorer, right-click on the Web reference and select Update Web Reference. Your new methods will be discovered, and you'll be instantly able to use the new features in code.

I'm wanting to expose some pretty sensitive information via my Web service. Do these things do security?

You bet your bottom dollar, kiddo. You can either implement your own authentication by adding username and password arguments to your methods, or use one of the ASP.NET authentication options, presenting your credentials before you begin using the service. For more information, look up "Web services, security" in the Help index for the full lowdown.

I like to know my file extensions. What does ASMX stand for?

Despite many leading publications revealing that "it doesn't have any real meaning," I can tell you that *ASMX* stands for *active server methods*, with the *X* just referring to the next generation of development techniques (that is, old ASP pages are now ASPX pages).

Good news! My Wonderful Weather Web service has grown to great heights. Bad news! My code runs a powerful algorithm to correctly predict whether it rains or shines, and that takes time. Now that I'm getting requests every second or so, my servers are clogging up. What can I do?

It's a little-known fact, but you can actually "cache" what a Web service gives out and automatically serve up the same response next time. How? Simply by specifying a CacheDuration in the WebMethod attribute, like this:

```
<WebMethod(CacheDuration:=60)> _
Public Function GetWeather() As String
     ' . . . complicated code . . .
End Function
```

Here, the Web server will cache and serve up the same results for GetWeather sixty seconds after the last query. You can test this baby by returning the time. See what happens?

Our company has just created a whole bundle of Web services containing company-wide functionality. The project is proving exceptionally popular among the developer team—and, to cope with demand, we're going to have to transfer the service to more powerful servers. Won't this break the programs currently using our Web service?

If your existing clients are looking to access a Web service and it isn't there, yup, their applications will either raise an error or die a dramatic death. Either way, it ain't good.

However, it's relatively simple to change the server name of your Web service. In Visual Studio .NET, simply open your project and select your Web service under the Solution Explorer, Web References folder. You should notice a few items in the Properties window—including Web Reference URL. Change this property to point to your new Service, then recompile your application.

If your Web service often changes location, it might be worthwhile changing the URL Behavior property to Dynamic. If you're working on a Web application, this adds a line containing the URL to Web.config (or App.config for Windows applications).

If changes occur in the future, you'll no longer have to redistribute all your files again—just send out an updated .config file. Top stuff!

WHERE TO GO FROM HERE

Looking to learn more about Web services? You're not alone—and, thankfully, the world is starting to brim with resources for the new technology. Helping you wade through the rubble, here's my pick of the best:

- *Architecting Web Services* (Apress, ISBN 1-893115-58-5): If you're looking for a full-on book delivering nothing but Web service information, this is the title for you. A real work of art.

- Other books: *Professional ASP.NET Web Services* (Wrox, ISBN 1-861005-45-8).

- Visual Studio .NET Help: The Visual Studio .NET documentation provides a full walkthrough of Web services in a data-driven environment. Definitely worth a browse. On the Programs menu, select Microsoft Visual Studio .NET ➢ Microsoft Visual Studio .NET Documentation.

- `www.xmethods.com`: Showcasing examples of Web services in action, this site is essentially the mini *Yahoo!* of the ASMX world. Current listings include an SMS service, a dictionary, and a horoscope service.

- `www.vbws.com/newsletter/`: Want the latest news on Web services? Trainer Yasser Shohoud publishes his own newsletter giving the full lowdown, each and every month. Subscribe for free here.

- `www.dotnetjunkies.com`: Incredibly useful .NET developer site, with pages dedicated to creating Web services. Tips, tricks, code samples, and more at this site. Definitely worth a surf.

- `www.asp.net`: Only Microsoft could own such an exclusive address. Lists all the latest books, community sites, and code samples.

- `www.vbxml.com`: Your one-stop shop for learning more about XML, the markup language behind Web services. Also includes numerous .NET code snippets.

- `www.webservices.org`: Providing industry news and details on Web services implemented on *all* platforms. On the whole, a pretty drab site—but, if you're into the likes of Linux and Java, a decent resource.

- `www.dnj.com`: Homepage of the popular Microsoft-endorsed programming magazine, *Developer Network Journal*. Regularly features articles on implementing Web service technologies. Plenty of online content too, plus the option to subscribe (plug, plug).

- `microsoft.public.dotnet.framework.aspnet.webservices`: The official Microsoft newsgroup for Web services discussion. For speedier answers, try searching the archives first at `www.googlegroups.com`.

CONCLUSION

When asked exactly what .NET is, Microsoft's official answer is always ".NET is the Microsoft Web services platform."

In other words, Web services are at the heart of Microsoft's vision for a distributed computing future. Yes, there's a lot of hype—but there's no denying that Web services are cooler than a Pepsi-drinking polar bear. And, over the last twenty or so pages, you've learned how to get your own up and running in minutes.

We launched the tutorial today by creating our own mini Web service, then looked at using its operations from a Web page—plus found out how to "discover" and "consume" that functionality from within our applications. After that, we talked about its possibilities, plus went on to access exceptionally "smart data" in under twenty lines of code.

Finally, we checked out deployment and the top five things you really should do before making your service available to the world. And finally-finally, we reviewed all those frequently asked questions alongside a list of resources to take your knowledge to the next level.

Just remember: the next time you need to get your computers more talkative than a Jerry Springer audience, think Web services. They get computers exchanging data, they distribute your application, they solve problems, they're just cool.

But that's all from me—until the next time, this is Lrak Eroom signing off for tonight, wishing you a very pleasant evening, wherever you are in the world. Goodnight!

CHAPTER 6.1

INTRODUCING THE WORLD OF WEB SERVICES REVIEW SHEET

- Web services allow you to get computers talking to each other. The process is often compared to the old DCOM; however, it is loosely coupled and based on open standards.

- On the server side of a Web service, you have chunks of compiled code that are exposed to the client. On the client side, you have an application or browser making HTTP requests to those code chunks. The .NET Framework passes data between requests about in XML (eXtensible Markup Language).

- To generate your own Web service, create a new ASP.NET Web service project in Visual Studio .NET. By default, you will have a Web service called Service1.asmx added to your application. You can add new Web services to a project by selecting Project ➢ Add Web Service from the menu.

- To author the code behind your Web service, right-click on the service in the Solution Explorer and select View Code.

- By default, none of your methods, functions, or properties are exposed as part of your Web service. To expose a chunk of code, prefix it with the `<WebMethod()>` attribute, as so:

```
<WebMethod()> Public Function GetDate() As Date
    Return Now
End Function
```

- You can view your Web service in a browser by simply building the project, then visiting its dynamically generated ASMX page. Another technique is to simply press the F5 key while working on your Web service project.

- To "discover" a Web service for use in your Visual Studio .NET application, select Project ➢ Add Web Reference from the menu. Specify the address of your Web service, press the Enter key, then select Add Reference. Alternatively, browse the Microsoft UDDI Directory at `http://uddi.microsoft.com/visualstudio`.

- To “consume” the functionality of a Web service in your application, simply treat it as you would any regular object. The XML is all parsed out and handled for you. For example, the following demonstrates a discovered Web service in use:

```
Dim MyObject As New LocalHost.Service1()
MessageBox.Show(MyObject.GetDate)
```

- Your application doesn’t care where it accesses Web services from. The data could be coming from the local computer, across a network, or from the other side of the world via the Internet. So long as that connection is available, the Web service works.

- To distribute your Web service to another server, from the menu select Project ➢ Copy Project. Alternatively, copy the ASMX, CONFIG, and VSDISCO files across to the new Web Application directory, plus create a subdirectory called Bin, copying your compiled Bin/ProjectName.DLL assembly into it.

- Web service members can accept and return more than just base data types. They can take in and pass back anything, including DataSets, data-holding classes, and user-defined types (structures).